

Elementi di programmazione

**Il progetto
dei programmi**

S.Fumich

1. LE FASI DELLO SVILUPPO DEL SOFTWARE.

1.0 Il processo di sviluppo del software e' un processo complesso che richiede un certo numero di fasi e di compiti particolari in ciascuna fase.

Lo sviluppo del software puo' essere sostanzialmente articolato nelle seguenti fasi:

1. definizione del progetto (cioe' di che cosa deve essere realizzato);
2. definizione dei requisiti (cioe' di cosa il nuovo software deve fare);
3. progettazione e programmazione (cioe' la determinazione di come il nuovo software lavorera' e la sua realizzazione);
4. verifica (cioe' la osservazione in ambiente controllato come il software realizzato lavora);
5. conversione (cioe' la valutazione dei tempi e la determinazione delle modalita' per l'inserimento del software nell'ambiente di utenza).

1.1 Nella prima fase, quella di definizione del progetto, si presentano ai progettisti diversi compiti, che possono essere cosi' riassunti:

1. identificazione degli utenti (sia quelli primari che quelli "remoti");
2. definizione dello scopo del progetto;
3. definizione delle limitazioni del progetto (in termini di tempo, denaro, personale a disposizione, ecc.);
4. definizione del tipo di progetto (il sistema dovra' soddisfare al meglio le necessita' dei suoi utenti, per cui vanno individuate incompatibilita' ed eventuali priorita');;
5. pianificazione del progetto (il piano sara' uno schema di come deve procedere lo sviluppo del progetto: tempi stimati di inizio e fine per ciascun compito particolare, proiezioni delle ore/uomo necessarie, ecc.).

1.2 Nella fase della definizione dei requisiti i compiti che si presentano ai progettisti sono:

1. definizione delle funzioni (per funzione si intende una unita' di lavoro che deve essere gestita dal software);
2. definizione dei dati (intervallo di validita', lunghezza massima di un dato, se numerico, alfabetico o alfanumerico, ecc.);
3. definizione delle restrizioni temporali per ogni funzione;
4. proiezioni di volume per ogni funzione.

1.3 I compiti della fase di progettazione e di programmazione si possono riassumere cosi':

1. progetto del data base;

2. progetto del processo (cioe' definizione di quali funzioni saranno automatizzate e quali eseguite manualmente);
3. controlli;
4. progetto del programma (cioe' realizzazione delle specifiche per le funzioni da automatizzare);
5. definizione delle procedure (per quelle funzioni destinate alla gestione manuale);
6. documentazione.

1.4 I compiti della fase di verifica sono:

1. definizione del piano di prova;
2. test con i dati di prova;
3. analisi dei risultati.

1.5 I compiti che vanno affrontati nella fase di conversione si possono elencare come segue:

1. piano di conversione;
2. introduzione dei nuovi file;
3. training del personale;
4. introduzione delle nuove procedure.

Il piano di conversione dovra' definire come installare il nuovo software. Sono possibili varie metodologie:

1. conversione parallela (il vecchio ed il nuovo software sono usati assieme con comparazione dei risultati; tale metodologia puo' essere costosa);
2. conversione a fasi (i singoli moduli sono pian piano implementati sostituendoli nell'ambito del vecchio software);
3. conversione pilota (viene usata in un sistema con molti utenti).

2. IL PROGETTO DEL PROGRAMMA.

2.0 La progettazione del programma e' un compito complesso che comprende diverse fasi, anch'esse complesse.

Per prima viene l'analisi del problema con la precisa determinazione e definizione degli inputs e degli outputs. Si passa quindi alla fase di progetto con la definizione dei moduli, la scrittura delle specifiche dei moduli, la preparazione dei dati di prova, la descrizione della logica e la revisione della logica stessa.

Seguono le fasi di codifica, di editing, il cosiddetto computer cycle, cioe' la compilazione, la eventuale correzione e ricompilazione. Si prosegue poi con un "giro a mano" strutturato, dopodiche' se c'e' la convinzione che il prodotto dovrebbe essere funzionante, si passa alla fase di verifica.

La verifica comprende la preparazione dei fogli con l'impostazione e i risultati previsti del test, il collaudo parziale dei moduli, il collaudo integrato del programma.

Dopo la verifica positiva si procede con l'ultima fase, cioe' la produzione di istruzioni operative (job control e istruzione operative per l'uso del programma).

2.1 Poiche' il processo di sviluppo di un programma e' un processo complesso, si deve usare per lo sviluppo di programmi un metodo organizzato.

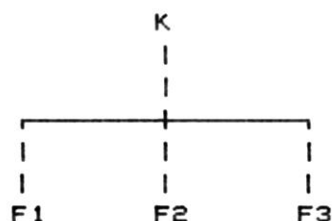
Il programma deve essere organizzato in modo tale da poter essere studiato e risolto in pezzi o "moduli" maneggevoli.

Un modulo e' una sezione di codice che esegue una funzione logica ben definita e facilmente isolabile (indipendente). Ha un nome, cosicche' puo' essere chiamato da altri moduli.

Un modulo puo' essere definito come una "scatola nera", con un input e un output, che esegue una trasformazione di dati con nessun effetto collaterale.

Mai il controllo passera' da un modulo nell'interno di un altro modulo.

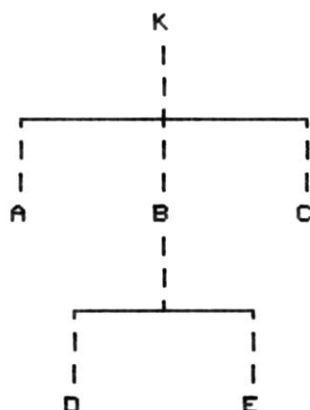
2.2 Una struttura ad albero puo' evidenziare la gerarchia dei moduli.



Il modulo in cima controlla l'intera struttura. E' il driver al quale fare riferimento: esso stabilisce quando ciascun modulo sara' eseguito.

Quando l'esecuzione di un modulo e' completata il controllo ritorna al modulo chiamante: mai il controllo deve essere passato a un altro modulo che non sia quello chiamante.

Nel seguente esempio, quando il driver chiama B, B puo' chiamare D e/o E. Se D e' chiamato, completata la sua esecuzione, il controllo ritorna a B ed infine al driver.

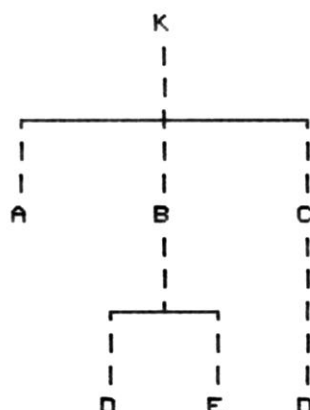


Un modulo controlla se' stesso piu' tutti i moduli ad esso subordinati.

Il campo di controllo del modulo principale e' l'intera struttura.

2.3 La struttura ad albero mostra quali moduli sono usati, non come sono implementati.

Nel seguente esempio, D e' usato da due moduli, B e C. D puo' essere del codice ridondante o puo' essere codificato una sola volta e catalogato come sottoprogramma.



2.4 Il flusso di programma all'interno di ciascun modulo sara'

limitato a pochi blocchi logici di base:

- sequenza
- IF-THEN-ELSE
- DO-WHILE-REPEAT
- DO-REPEAT-UNTIL.

3. STRUMENTI DI PROGETTO.

3.0 Flowcharts, tavole di decisione, strutture ad albero, pseudocodice ed altri ancora sono gli strumenti comunemente usati nel processo di sviluppo del software.

La scelta di uno di tali strumenti va valutata in termini di capacita' di implementazione delle strutture base di controllo.

Ci occuperemo qui brevemente dei primi tre tipi di strumenti elencati, facendo gia' buon uso del quarto sin dall'inizio del corso.

3.1 L'uso di flowcharts in un ambiente strutturato e' generalmente scoraggiato in quanto essi danno al programmatore la capacita' di creare una struttura incomprensibile.

"It furnishes the programmer with a tool for creating a monster; the ending of every monster story is always the same: the monster destroys its creator. Unfortunately, in the data processing world it is not just the creator but often the entire organization that is destroyed."
(G. Harrington Swann)

Tuttavia con disciplina e controllo con il flowchart si puo' lavorare molto bene in un ambiente strutturato.

Cioe', il programmatore deve trattare i flowcharts in pezzi maneggevoli, in maniera modulare, in modo che ciascun pezzo possa essere studiato e risolto separatamente.

3.2 Le tavole di decisione mostrano sulla base di un insieme di condizioni quale elaborazione ha luogo. La sequenza di processo puo' differire a seconda delle condizioni verificate.

Con esse si lavora molto bene quando si ha a che fare con scelte, mentre risulterebbe al contrario molto scomodo mostrare con esse una sequenza o un ciclo.

3.3 Le strutture ad albero sono uno strumento efficace per mostrare a grandi blocchi come e' costituito un sistema, per riassumerlo in termini di "cosa si fa" piuttosto che di "come si fa".

Per quanto riguarda la traduzione di sequenze si conviene che i moduli di pari livello sono eseguiti da sinistra a destra, mentre vengono adottati speciali simboli per indicare un test per determinare l'esecuzione alternativa di moduli, o per indicare processi iterativi coinvolgenti i moduli.

4. IL PIANO DI LAVORO.

4.0 Si puo' dire innanzitutto che e' importante che il piano di lavoro sia tagliato per l'organizzazione particolare che sviluppera' il progetto. E' estremamente importante che ogni persona capisca ciascun passo del processo di sviluppo.

Il piano di lavoro deve quindi definire chiaramente che cosa si deve fare.

Il piano di lavoro dirigera' l'operato dei progettisti durante le varie fasi che abbiamo gia' individuato. Ad esempio, per la fase di progettazione e programmazione una scaletta a grandi blocchi potra' dunque essere la seguente:

- analisi del problema
- progetto
- codifica
- editing
- computer cycle
- "giro a mano"
- collaudo
- produzione di istruzioni operative.

Affronteremo qui nel seguito la discussione di alcuni dei punti elencati, iniziando dal secondo (la fase di progetto) ed in particolare dalla definizione dei moduli.

4.1 La definizione dei moduli e' l'essenza stessa della programmazione strutturata ed e' la cosa piu' difficile da fare.

Non esistono regole precise per ottenere la definizione migliore, ma si possono dare indicazioni e tecniche che possono essere usate con successo.

Il primo passo sara' quello di listare a grandi blocchi le funzioni, cioe' il lavoro che deve essere fatto dal programma. Quindi si progetteranno almeno tre differenti strutture ad albero che contengano tali funzioni. Infine si decidera' quale di tali strutture viene maggiormente incontro ai nostri obiettivi particolari.

Le tre differenti strutture possono non essere il miglior progetto, ma cosi' almeno si ha un punto concreto di partenza, certamente migliore di un foglio completamente bianco.

4.2 E' opportuno sin dall'inizio decidere approssimativamente quante linee di codice formeranno ciascun modulo, tenendo conto delle seguenti osservazioni:

1. il numero di linee di codice puo' essere un indicatore della complessita' del modulo;

2. non si può considerare, tuttavia, la lunghezza come una regola fissa;
3. moduli con poche linee di codice (meno di 10) possono non eseguire una funzione completa;
4. moduli con troppe linee (100 o più) possono fare troppo (si può cioè considerare di spezzare il modulo in sottomoduli);
5. un modulo può contenere più di 100 linee di codice ma può essere semplice e facilmente comprensibile;
6. una struttura con troppi moduli non è desiderabile (il controllo dell'esecuzione di questi moduli può essere troppo complicato).

Una regola da seguire è definire moduli che producano programmi comprensibili, leggibili, modificabili e sicuri. La lunghezza è un criterio importante per definire i moduli, ma non è il solo.

4.3 Bisogna controllare accuratamente la lista delle funzioni in modo da essere sicuri che sia completa.

Il progetto iniziale va quindi analizzato tenendo conto che è senz'altro desiderabile far sì che ciascun modulo svolga una funzione completa ed indipendente, senza curarsi della lunghezza, ma che tuttavia ciò può causare una struttura ad albero "troppo caricata di lavoro" che può creare problemi a seconda di come è implementata (troppi accessi a disco, troppe chiamate di routine).

Pertanto, è opportuno cercare di minimizzare per quanto è possibile le ramificazioni (ad esempio, mettendo assieme moduli che sono correlati logicamente).

Inoltre è opportuno determinare se qualche funzione è usata in qualche altro programma (se è così è buona cosa costruirla in un modulo separato cosicché può essere usata da altri programmi).

4.4 Nel definire i moduli bisogna tener conto del controllo. I moduli comunicheranno l'uno con l'altro? Ci dovranno essere dei flag da passare da un modulo all'altro per far sì che la struttura lavori?

I flag o gli switch sono una fonte di buchi e causano dipendenza tra moduli.

Il campo di effetto di una decisione deve essere il campo di controllo del modulo contenente la decisione.

Bisogna mirare a creare moduli indipendenti. I più grandi risparmi di costi sono realizzati quando i moduli possono essere capiti e risolti indipendentemente da qualunque altro modulo nella struttura.

4.5 Nell'ambito di un progetto bisogna considerare eventuali

politiche e standard d'azienda che possono influenzare il progetto. Ad esempio, l'azienda può avere la politica che tutti gli input e gli output devono essere in un modulo separato.

4.6 In un modulo veramente coerente ogni pezzo di codice è connesso con l'esecuzione di una funzione.

Per quanto riguarda la coesione possiamo classificare i metodi di divisione di un programma in moduli, secondo una scala di valore che va dalla soluzione meno desiderabile a quella più desiderabile, come segue:

1. moduli coincidenti (nessuna relazione significativa tra elementi in un modulo: i moduli sono stati divisi arbitrariamente);
2. moduli logici (ciascuna chiamata esegue una di una classe di funzioni connesse, ad esempio un modulo che gestisce tutti gli input e gli output);
3. moduli classici (elementi delimitati logicamente sono eseguiti sequenzialmente nel tempo, ad esempio un modulo che gestisce tutte le inizializzazioni);
4. moduli funzionali (tutti gli elementi sono connessi per eseguire una singola funzione).

4.7 Occupiamoci ora della definizione delle specifiche dei moduli.

Scrivere le specifiche per ciascun modulo permette di lavorare su un singolo modulo alla volta, e di effettuare facilmente modifiche al programma senza preoccuparsi di verificare quegli elementi non connessi col modulo da modificare.

Il foglio con le specifiche di un modulo contiene:

1. l'indicazione di come è eseguito, se serialmente o iterativamente;
2. la lista di dati di input necessari con il nome mnemonico che deve essere usato dal programmatore (i dati sono classificati dal tipo, ad esempio C = dati di controllo, P = dati da elaborare; per ciascun elemento va indicato se è una costante o una variabile. Inoltre è buona cosa non dimenticare che switch e flag possono causare problemi quando i moduli sono modificati, e quindi limitarne l'uso);
3. la lista dei dati in output con l'indicazione se sono dati di input usati dal modulo e passati senza modifiche, dati di output finali passati al sistema e non usati dai moduli seguenti, dati infine il cui valore è modificato dal modulo.

4.8 Per quanto riguarda la preparazione dei dati di prova, va sottolineato che i dati di prova vanno preparati usando il foglio delle specifiche del modulo prima di iniziare la codifica.

La preparazione preventiva dei dati di prova permette di chiarire potenziali problemi di codifica, come anche di verificare la propria comprensione del problema.

Se non si hanno abbastanza informazioni per preparare i dati di prova, non si possono avere sufficienti informazioni per sviluppare la logica di quel modulo.

4.9 Sulla descrizione della logica ricorderemo soltanto il principio fondamentale, essendoci dilungati sull'argomento già in altra parte del corso: si parte con una soluzione a grandi blocchi del problema e si procede via via fino ai dettagli.

4.10 I diagrammi logici o lo pseudocodice devono essere rivisti prima di iniziare la codifica.

Un "giro a mano" della logica serve a scoprire errori nella logica, a scoprire specifiche incomplete per una funzione, come esperienza didattica perché i programmatori possono vedere come un altro ha risolto una particolare applicazione.

Se viene trovato un errore, sarà opportuno un ulteriore "giro" prima di passare alla codifica.

La codifica comincia solo quando ognuno è sicuro che la logica sia corretta.

Nella codifica saranno usati:

- soltanto i blocchi logici di base;
- moduli di 50-60 linee di codice;
- uno statement per linea;
- identificatori per mostrare i corrispondenti IF ed ELSE;
- commenti per spiegare i moduli e qualche logica particolarmente complessa;
- nomi di dati e di paragrafi significativi;
- GO TO solo per andare ad un exit point.

4.11 Ancora un "giro a mano" va fatto dopo aver eliminato gli errori di compilazione. Esso può assicurare che la logica revisionata in precedenza è stata codificata correttamente; può costituire una verifica che non vi sono violazioni di standard di codifica; infine può dare la certezza che indici e switch sono definiti in maniera efficiente ed usati propriamente.

4.12 Per quanto riguarda la fase di testing daremo qui soltanto alcune indicazioni, in quanto l'argomento verrà ripreso in altra parte del corso.

Ogni modulo di programma va testato separatamente. Nella prova saranno inclusi anche moduli testati in precedenza.

Se vi sono moduli chiamati non ancora codificati o verificati, sarà necessario simulare la loro presenza con messaggi che indichino il pieno successo della chiamata.

5. L' USO DI LOGICA PREPROGETTATA.

5.0 Uno degli obiettivi principali della programmazione strutturata e' quello di aumentare la produttivita' dei programmatori. Un metodo per raggiungere tale obiettivo e' l'uso di programmi preprogettati.

Molti problemi hanno la stessa base logica: quando un programmatore capisce le funzioni logiche di base, i programmi possono essere codificati piu' velocemente e piu' accuratamente, ed inoltre viene ridotto il tempo necessario per il test ed il debug dei programmi.

5.1 Ad esempio, la maggior parte delle applicazioni commerciali possono essere catalogate come insiemi di una o piu' delle seguenti sette funzioni base: EXTRACT, SUMMARIZE, UPDATE, MATCH, MERGE, LOOK-UP, SORT.

I significati di tali funzioni possono essere cosi' riassunti:

EXTRACT - Ciascun record su un file e' elaborato senza riferimento ad un qualunque altro record sul file.

SUMMARIZE - Si deve controllare il record corrente in funzione del precedente. Prima che un record sia elaborato, esso deve essere comparato, usando un campo di controllo, con il record precedente. Il risultato della comparazione e' usato per controllare speciali processi che sono eseguiti soltanto quando ci sono cambiamenti nei vari livelli del campo di controllo (sommari, totali, tabulazioni, cross-tabulations, rapporti gerarchici, ecc.).

UPDATE - Records da un file di input (transaction file) sono letti e comparati su campi di controllo con la conseguenza di cambiare, cancellare o aggiungere records su un altro file (master file) a seconda delle informazioni in essi contenute.

MATCH - Records sono letti da files di input e comparati su campi di controllo in modo da poter elaborare ciascun record del primo file (detail file) con un record corrispondente, se c'e', dell'altro file (master file).

MERGE - Sono letti records da files di input, sono comparati su un campo di controllo e uniti per fare un unico file contenente tutti i record. Il file in output e' nella sequenza dei files di input.

LOOK-UP - Records su files di input non sono nella stessa sequenza. Ciascun record su un file ("file") e' letto e comparato su un campo di controllo con records dell'altro file ("table") finche' non e' trovato il record corrispondente. Il record di "file" e' elaborato con il record "table" se trovato, altrimenti senza.

SORT - Records di un file di input sono messi in sequenza per mezzo di

un campo di controllo.

6. L'ASPETTO UMANO.

6.0 L'approccio strutturato si presta bene al lavoro di team. Con i moduli definiti e le specifiche scritte per ciascun modulo, il modulo puo' essere codificato simultaneamente dai membri di un team.

Un team puo' consistere di:

- un team leader (programmatore capo), che definisce i moduli, scrive le loro specifiche, fa da guida per gli altri membri del team, possiede capacita' di analisi e tecniche di programmazione;
- un librarian, che prepara i dati di prova, cura la documentazione, stabilisce i nomi dei dati per i programmatori, fa la compilazione, corregge gli errori di editing, gestisce i lanci di prova;
- membri, che aiutano a definire i moduli, codificano i moduli, aiutano a sviluppare i dati di prova;
- sostituto, che deve essere qualificato per gestire qualunque compito in modo da poter continuare il lavoro se un membro del team non e' disponibile.

Il lavorare o meno in team e' ovviamente in funzione della mole del progetto.